(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 May 2002 (02.05.2002)

PCT

(10) International Publication Number
WO 02/35349 A1

(51) International Patent Classification[7]: G06F 9/45

(21) International Application Number: PCT/US01/31713

(22) International Filing Date: 11 October 2001 (11.10.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/697,061 26 October 2000 (26.10.2000) US

(71) Applicant *(for all designated States except US)*: **VIRTUAL MEDIA, INC.** [US/US]; 18-3 East Dundee Avenue, Suite 300, Barrington, IL 60010 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **UNER, Eric, R.** [—/US]; 1822 Cambridge Drive, Carpentersville, IL 60110 (US).

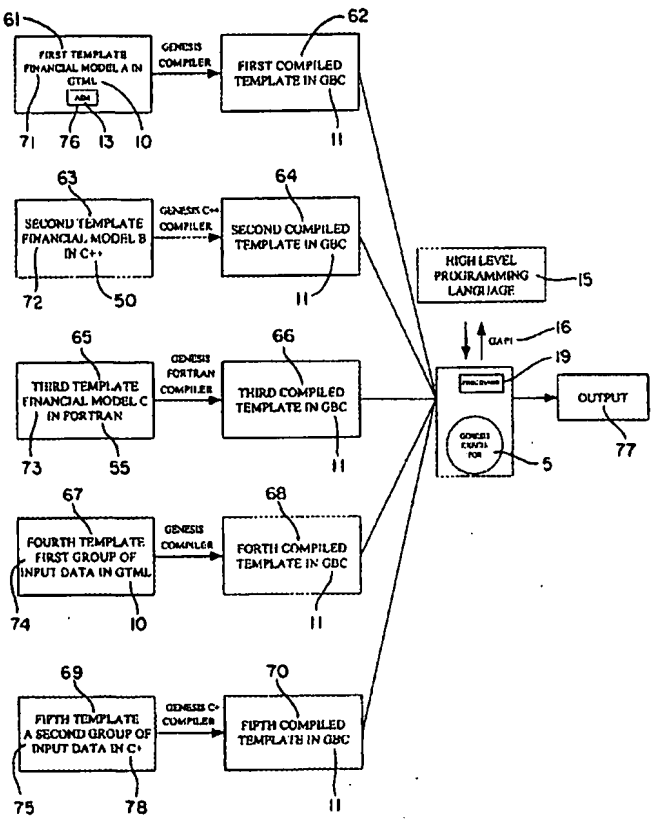
(74) Agent: **MERONI, Charles, F., Jr.**; P.O. Box 309, Barrington, IL 60011 (US).

(81) Designated States *(national)*: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF,

[Continued on next page]

(54) Title: TRANSLATING DATA STREAMS USING INSTRUCTIONS IN TEMPLATES



(57) Abstract: This invention is system for translating data streams. The method comprises programming pre-arranged instructions into a first template (61, fig. 6) in a Genesis Template Markup Language (GTML), compiling the first template into Genesis Byte Code (GBC) (62), creating a second template in the GTML (63), compiling the second template into GBC (64), interpreting the first and the second compiled templates to a processor by a Genesis Executor (GE) (5), and generating output data (77). The apparatus comprises a first compiler on a first system for compiling a first template (61) in a first high level language into GBC (62), a second compiler on a second system for compiling a second template (63) in a second high level language into GBC (64), a GE (5) on a third system for providing an interpretation of the first and second compiled templates, and a processor (19) on a forth system for generating output data (77) based on the GE (5).

WO 02/35349 A1

CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *with international search report*

SPECIFICATION

TRANSLATING DATA STREAMS USING INSTRUCTIONS IN TEMPLATES

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to data processing methods, apparatus and computer program products. More specifically, my invention is primarily intended for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates.

Description of the Prior Art

Most digital devices, including computers, embedded systems, and like, require a program to help them operate. A program is a specific set of ordered operations for a device to perform. These devices would benefit greatly from separating the core functions of their program from the more specific portions of their program. For example, a digital watch benefits from having one part of its design handle the actual timekeeping, while another part handles the display of the time. In this way, the maker of the watch can use the core time keeping functionality in many different watches with many different displays.

This concept of separating functionally unique parts of a design is important in a good engineering design, and is often called modularization. Software is usually

modularized by placing related instructions into separated files or groups of files. The files, called source files, contain instructions written in a high level computing language, such as C++, Java, Perl, or like. These files are then typically compiled into a binary code specific to the machine they are to be executed on. Some systems allow for an additional step that can translate data and text of different languages into a common language, where they can be compiled together into a single executable program. Some languages allow the system to bypass the compilations steps entirely, and execute the source files directly by interpreting their instructions real-time. Examples of interpreted languages include UNIX Shell Scripts, Perl, AppleScript, or like. Occasionally, interpreted languages allow a pre-compile step to produce a simple version of the source file that is easier for the interpreter to execute. An example of this would be Java. Java can compile into a form that is mostly platform independent, and be executed by a "Virtual Machine" or interpreter specific to a device or type of device.

In order to achieve the modularization, various inventions have been made. U.S. Patent No. 5,546,583, which issued to Shriver, discloses a Method and System for Providing a Client/Server Interface in a Programming Language. This invention provides a method and system for providing a client/server interface to a programming language. This invention further provides a method and system for providing client/server support which permits a server to wait for client requests and to read and set client variables. This invention also provides the user of a programming language a method and system for providing client/server support that allows users to benefit from the programming language's friendly "look and feel." In a data processing system, a programming language processor capable of executing program code is provided. A client program and a server

program are also provided within the data processing system. The client program and the server program are comprised of program code capable of execution within the data processing system. Once the client and server programs are invoked, the client program sends a request for a service to the server program. In response to program code within the server program, a request is sent to the client program for a service that requires access to a variable within the client program. The client program then processes the request from the server program and sends the server program a response. Thereafter, the server program continues processing the request from the client program in response to gaining access to the variable in the client program. If the server program has not been initialized when the client program requests a service, the client program automatically initializes the server program.

United States Patent No. 5,956,709, which issued to Xue, discloses a Dynamic Data Assembling on Internet Client Side. This invention eliminates the unnecessary transmissions of useful data by implementing all edit operations, such as adding, deleting, updating, entry check, calculation, and backup on client side in a data assembling process. This invention also reduces the time needed to assemble a data set needed to make a transaction on the Internet for Internet users. This invention further makes an online transaction a pleasant experience for Internet users instead of a tedious and time-consuming activity. The invented DDAICS method is an alternative diverse type of dynamic information retrieval, storage, and exchange method. Unlike those conventional Internet data retrieval and exchange methods on the Internet in which most application logic and data access logic are located on the server side, the newly invented DDAICS method is to arrange application logic and data access logic on both client side and server

side according to execution efficiency to achieve optimal implementations of transactions. A web browser with a build-in engine that can execute client side application program, such as Netscape Navigator 3.0 or above, is needed to implement the method. Web pages used in the method are specifically designed in which client side application program, such as one written with JavaScript, is embedded. Also a frame in a window, which is preferred, or a new window is created on client side as a data assembling monitor window. In addition, data needed for a transaction in a web page must be designed in such a way that they can be retrieved dynamically and individually and assembled with necessary user input in the monitor window by the client side application program actuated by users. In an assembling process, all or some of editing operations, such as adding, deleting, updating, entry check, calculation, and backup, are executed on client side until final submission, save, or print. The philosophy behind the DDAICS method is that where execution efficiencies lie, where application logic and data access logic go.

United States Patent No. 5,961,586, which issued to Pederson, discloses a System and Method for Remotely Executing an Interpretive Language Application. This invention relates to a method for remotely executing interpretive languages in a client-server environment. The server to which a client is connected downloads and executes an application written in an interpretive language, such as a JAVA applet. The server accepts input from, and provides screen data to, the client. This allows the client to appear as if it is executing the application in a traditional manner without requiring the client to expend compute and memory resources hosting and executing the application. Additionally, the server may be able to download the application more quickly than the

client. The server also accepts input from the client node, allowing the client node to control and provide input to the downloaded application. In one aspect, this invention relates to a method for remotely executing an application written in an interpretive language which begins by downloading the application to a server node in response to a request made by a client node. A connection is established between the client node and a predetermined communications port located on the server; the server creates an endpoint data structure and associates a client space hosted by the server with the endpoint data structure. The server generates a protocol stack associated with the client space and the associated endpoint data structure, notifies a connection manager of the connection, and transfers the connection between the predetermined communications port and the client node to the associated protocol stack. In another aspect, this invention relates to an article of manufacture having computer-readable program means embodied thereon for remotely executing an application written in an interpretive language. The article of manufacture includes: computer-readable program means for downloading the application to a server node in response to a request made by a client node; computer-readable program means for establishing a connection between the client node and a predetermined communications port located on the server; computer-readable program means for creating an endpoint data structure; computer-readable program means for associating a client space hosted by the server with the endpoint data structure; computer-readable program means for generating a protocol stack associated with the client space and the associated endpoint data structure; computer-readable program means for notifying a connection manager of the connection; and computer-readable program means for transferring the connection between the predetermined communications port

and the client node to the associated protocol stack. In still another aspect, this invention relates to a system for remotely executing an application written in an interpretive language. The system includes a server node having a predetermined communications port and a client node having a communications device establishing a connection between the client node and the predetermined communications port of the server node. A protocol stack is located on the server node and the protocol stack includes an endpoint data structure. A client space located in memory on the server node is associated with the protocol stack and provides an execution environment for an application written in an interpretive language. The system further includes a communication manager located on the server node, and a notification device located on the server node. The notification device notifying the connection manager of the connection between the client node and the predetermined communications port and the communications manager transferring the connection between the predetermined communications port and the client node to the protocol stack.

United States Patent No. 5,987,480, which issued to Donohue, discloses a Method and System for Delivering Documents Customized for a Particular User Over the Internet Using Imbedded Dynamic Content. This invention provides the ability to embed dynamic content into web pages. A system and method for delivering documents having dynamic content embedded over the worldwide Internet or a local internet or intranet. A data source is stored on a server computer connected to the Internet, the data source containing content in a form representing or reducible to names and corresponding values. Document templates are created by embedding dynamic tags and flow directives in markup language documents, the dynamic tags and flow directives containing one or

more names of content stored in the data source. The document templates are stored on the server computer. The server computer can receive requests from client computers connected to the Internet, the requests identifying desired documents to be delivered. In response to such a request, the server computer selects one of the document templates corresponding to the desired document, populates the document template with content stored in the data source based on respective values of content corresponding to names in the dynamic tags and flow directives, and delivers the populated document to the client computer. The invention has particular application to HTML documents transferred over the World Wide Web.

United States Patent No. 6,073,163, which issued to Clark, discloses a Method and Apparatus for Enabling Web-based Execution of an Application. This invention provides a method and system for executing an application at a client. According to the method, a first portion of code for the application is transmitted to the client over a network. The first portion of code is executed at the client to cause the client to generate a user interface that is displayed by the client, detect user interaction with the user interface, and transmit from the client over the network a first message that indicates the user interaction with the user interface. A second portion of code for the application is executed at a node of the network other than the client. The second portion of code causes the node on which it is executing to respond to the first message by (1) generating, based on the user interaction, at least one record that indicates a change to the user interface, and (2) transmitting the at least one record to the first portion of code in a second message. The first portion of code responds to the second message by performing the change to the user interface. According to one aspect of the invention, the step of

generating at least one record includes the step of generating a plurality of records which specify a plurality of changes to the user interface. The plurality of changes are accumulated by the second portion of code prior to transmitting the second message. The plurality of records are then sent in the second message.

United States Patent No. 6,078,322, which issued to Simonoff, discloses a Methods permitting rapid generation of platform independent software applications executed on a universal client device. This invention provides a computer system for interconnecting various military components efficiently. According to one aspect of this invention, the computer system advantageously permits military components to use the same computer program and share information beyond the visualization of a map, text or photograph regardless of variations in hardware and software between the networked computers. According to another aspect of the invention, a dedicated scripting language enables each military component to quickly and easily personalize the user front end, which presents the GUI objects, without modifying the same software program application used by all networked military components. Thus, the Government simultaneously achieves military component interoperability and cost savings regardless of computer variation and architecture. This invention also provides a computer system whereby research scientists designing systems employing simulation-based design technology are permitted to run simulations and visualize the results regardless of computer variation. According to one aspect of this invention, the computer system according to the present invention beneficially permits geographically dispersed users to access a central database, to run simulations, and to receive simulation results. According to yet another aspect of the present invention, the received simulation results

advantageously are displayed as directed by the user. This invention further provides a device which advantageously enables application programmers to quickly and easily script application program behavior without requiring modification to the device. This invention further provides an interface development method which advantageously enables application programmers to quickly and easily script application program behavior without requiring concurrent modification to the application program.

Although most of these mechanisms allow for modularization at the design level, none of them allow that flexibility at the system level. No matter which application development architecture is chosen, one or more challenges still exist. The first challenge is that building all functions into one application requires rebuilding the entire application for each change. Even a small change in any part of the application may require a complete rebuild and reinstall of the entire application. For example, making a button on a web page say "Please Submit" as opposed to simply "Submit" would require such a monumental task, which is an inordinate amount of effort for such a small change. The situation is complicated by both large enterprise environments and small embedded environment. Embedded systems are sometimes built into a microchip, and so a small change would require redesigning and manufacturing a new chip. Some embedded systems, such as cell phones and pagers, are so large in the number of units that distributing any kind of alteration or upgrade to them that requires physical possession of the device is simply not practical. In enterprise environments, the mere act of building the software may take hours or even days. In some systems, restarting the system even after the build would require too much downtime of the system to justify the change.

The second challenge is that the software developers are required to have good user interface skill in devices that feature a user interface. Typically, designers and artists may have good computer sills, but little or no programming skills. The converse is true for programmers, who often have little or no design experience. Cases where an individual has acquired both skill sets are extremely rare, and it is even more rare for that individual to perform the task with the same competency as a specialist in the field. This rift in the talent base creates a need for programmers and designers to work hand in hand, with one side's work and schedule necessarily affecting the other's. For example, a programmer working on a web site would need to know the design of the web page in any places where the software must generate dynamic content. A delay in the approval of the design of the web page may delay the software development.

The third challenge is that interpreted languages are typically much slower than compiled applications. An interpreted program usually cannot run nearly as fast as a compiled program. This is due to the fact that in a compiled language, a language compiler converts source statements into something close to the strings of 0's and 1's that a processor ultimately is given to work on. Because this work must be done on an interpreted language at the time the interpreted program is run, there is an extra step involved in the execution that ultimately slows the program down.

The fourth challenge is that real time interpreters often have no communication with the core application. In environments that mix compiled applications with interpreted scripts, program instructions and data in each are separate, with no opportunity to share without storing the data in a temporary location common to both sides.

The fifth challenge is that placing all code in interpreted source files makes all code visible. Having no way to compile the code makes obfuscation difficult if not impossible. For example, in an interpreted script that accesses a database with a password, that password must be included in the script. Compiling that same script would hide the password and along with any algorithms in that script.

Because of all these challenges, what is needed then is a method, apparatus and computer program product for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates.

Accordingly, it is a principal object of my invention to provide a method, apparatus and computer program product for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates.

It is a further object of my invention to provide a method, apparatus and computer program product that separates an operating program into two independent parts, one is a model template, the other is an input data template.

It is a still further object of my invention to provide a method, apparatus and computer program product that enables a model to be used by the other parties without disclosure any of the content of the model.

It is a further object of my invention to provide a method, apparatus and computer program product that enables a programmer to program a model without a need to know the design of the user interface.

It is a still further object of my invention to provide a method, apparatus and computer program product that enables a user interface designer to design a user interface without a need to know the content of the model.

It is a further object of my invention to provide a method, apparatus and computer program product that uses Genesis Executor to integrate the model and the template to generate an output.

It is a further object of my invention to provide a method, apparatus and computer program product that enables the separation of the processing and translation of data from the core device or application.

Other objects of my invention, as well as particular features, elements, and advantages thereof, will be elucidated in, or apparent from, the following description and the accompanying drawing figures.

SUMMARY OF THE INVENTION

According to my present invention I have provided a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. This method comprises programming a group of instructions into a first template, the group of instructions can be either a group of instructions or a group of instructions with associated data, compiling the first template into a first compiled template in Genesis Byte Code through a first compiler, creating a second template with a group of input data, the group of input data can either be a group of data or a group of data with associated instructions, compiling the second template into a second compiled template in Genesis Byte Code through a second compiler, interpreting the first compiled template and the second compiled template to a processor by a Genesis Executor, and generating a group of output data by the processor.

The first template is in a first high level programming language, the first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, the Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, the Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, the Genesis Application Program Interface is a feature rich set of functions that allow the another high level programming language to pass data back and forth to a current execution of both the first compiled template and the second compiled

template, the another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

The second template is in a second high level programming language, the second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, the Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, the Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, the Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both the first compiled template and the second compiled template, the another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

Either one or both of the first template and the second template can include a group of data in assembly language. Either one or both of the first template and the second template can be a single template or a group of templates. Either one or both of the first compiled template and the second compiled template can be a single compiled template or a group of compiled templates. Either one or both of the first template and the second template may contain either text, or data, or instructions, or any combination of text, data, and instructions. The first template and the second template are created independently from each other.

The first compiler is a utility that takes the first template in the first high level programming language as input, and produces the first compiled template in the Genesis

Byte Code as output. The second compiler is a utility that takes the second template in the second high level programming language as input, and produces the second compiled template in the Genesis Byte Code as output.

The first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the first compiled template. The second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the second compiled template. Both the first compiled template and the second compiled template contain the Genesis Byte Code suitable to be interpreted by the Genesis Executor to the processor, along with a set of maps to variables, constants, and code required by the Genesis Byte Code.

The Genesis Byte Code consists of an ordered set of binary representations of instructions. The Genesis Byte Code is the lowest level form of code used by the Genesis Executor. The Genesis Byte Code is also a binary representation of the instruction mnemonics created by either the first compiler or the second compiler.

According to my present invention I have also provided an apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The apparatus comprises means for programming a group of instructions into a first template, the group of instructions can be either a group of instructions or a group of instructions with associated data, means for compiling the first template into a first compiled template in Genesis Byte Code through a first compiler, means for creating a second template with a group of input data, the group of input data

can either be a group of data or a group of data with associated instructions, means for compiling the second template into a second compiled template in Genesis Byte Code through a second compiler, means for interpreting the first compiled template and the second compiled template to a processor by a Genesis Executor, and means for generating a group of output data by the processor.

The first template is in a first high level programming language, the first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, the Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, the Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, the Genesis Application Program Interface is a feature rich set of functions that allow the another high level programming language to pass data back and forth to a current execution of both the first compiled template and the second compiled template, the another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like. The first template and the second template are created independently from each other.

The second template is in a second high level programming language, the second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, the Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, the Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application

Program Interface, the Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both the first compiled template and the second compiled template, the another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

Either one or both of the first template and the second template can include a group of data in assembly language. Either one or both of the first template and the second template can be a single template or a group of templates. Either one or both of the first compiled template and the second compiled template can be a single compiled template or a group of compiled templates. Either one or both of the first template and the second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

The first compiler is a utility that takes the first template in the first high level programming language as input, and produces the first compiled template in the Genesis Byte Code as output. The second compiler is a utility that takes the second template in the second high level programming language as input, and produces the second compiled template in the Genesis Byte Code as output. The first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the first compiled template. The second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the second compiled template. The first compiled template and the second compiled template contain the Genesis Byte Code suitable to be interpreted by the

Genesis Executor to the processor, along with a set of maps to variables, constants, and code required by the Genesis Byte Code.

The Genesis Byte Code is the lowest level form of code used by the Genesis Executor. The Genesis Byte Code consists of an ordered set of binary representations of instructions. The Genesis Byte Code is a binary representation of the instruction mnemonics created by either the first compiler or the second compiler.

According to my present invention I have further provided a computer program product recorded on a computer readable medium for a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The computer program product comprises computer readable means for programming a group of instructions into a first template, the group of instructions can be either a group of instructions or a group of instructions with associated data, computer readable means for compiling the first template into a first compiled template in Genesis Byte Code through a first compiler, computer readable means for creating a second template with a group of input data, the group of input data can either be a group of data or a group of data with associated instructions, computer readable means for compiling the second template into a second compiled template in Genesis Byte Code through a second compiler, computer readable means for interpreting the first compiled template and the second compiled template to a processor by a Genesis Executor, and computer readable means for generating a group of output data by the processor.

The first template is in a first high level programming language, the first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, the Genesis Template Markup Language

features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, the Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, the Genesis Application Program Interface is a feature rich set of functions that allow the another high level programming language to pass data back and forth to a current execution of both the first compiled template and the second compiled template, the another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like. The first template and the second template are created independently from each other.

The second template is in a second high level programming language, the second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, the Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, the Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, the Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both the first compiled template and the second compiled template, the another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

Either one or both of the first template and the second template can include a group of data in assembly language. Either one or both of the first template and the second template can be a single template or a group of templates. Either one or both of

the first compiled template and the second compiled template can be a single compiled template or a group of compiled templates. Either one or both of the first template and the second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

The first compiler is a utility that takes the first template in the first high level programming language as input, and produces the first compiled template in the Genesis Byte Code as output. The second compiler is a utility that takes the second template in the second high level programming language as input, and produces the second compiled template in the Genesis Byte Code as output. The first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the first compiled template. The second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the second compiled template. Both the first compiled template and the second compiled template contain the Genesis Byte Code suitable to be interpreted by the Genesis Executor to the processor, along with a set of maps to variables, constants, and code required by the Genesis Byte Code.

The Genesis Byte Code is the lowest level form of code used by the Genesis Executor. The Genesis Byte Code consists of an ordered set of binary representations of instructions. The Genesis Byte Code is a binary representation of the instruction mnemonics created by either the first compiler or the second compiler.

DESCRIPTION OF THE DRAWINGS

Other features of my invention will become more evident from a consideration of the following detailed description of my patent drawings, as follows:

Figure 1 is a first embodiment of this invention having a financial model in GTML and input data in GTML;

Figure 2 is a second embodiment of this invention having a financial model in GTML with a group of data in ASM and input data in GTML;

Figure 3 is a third embodiment of this invention having a financial model in GTML and input data in GTML with a group of data in ASM;

Figure 4 is a fourth embodiment of this invention having a financial model in GTML with a group of data in ASM and input data in GTML with a group of data in ASM;

Figure 5 is a fifth embodiment of this invention having a financial model in C++ and input data in FORTRAN;

Figure 6 is a sixth embodiment of this invention having three different financial models and two input data templates;

Figure 7 is a seventh embodiment of this invention having three different groups of instructions and two input data templates;

Figure 8 is an eighth embodiment of this invention having two groups of instructions and five input data templates;

Figure 9 is a first system on which this invention can be practiced; and

Figure 10 is another system on which this invention can be practiced.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 describes a first embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a financial model 9 into a first template 1 in a Genesis Template Markup Language (GTML) 10; a step of compiling the first template 1 into a first compiled template 2 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of creating a second template 3 in the Genesis Template Markup Language (GTML) 10 with a group of input data 12; a step of compiling the second template 3 into a second compiled template 4 in the Genesis Byte Code (GBC) 11 through the Genesis Compiler 8; a step of interpreting the first compiled template 2 and the second compiled template 4 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 7 by a processor 19. Either one or both of the first template 1 or the second template 3 can be a single template or a group of templates. Either one or both of the first template 1 and the second template 3 may contain either text, or data, or instructions in the Genesis Template Markup Language (GTML) 10, or any combination of text, data, and instructions in the Genesis Template Markup Language (GTML) 10. Either one or both of the first compiled template 2 and the second compiled template 4 can be a single compiled template or a group of compiled templates. The Genesis Compiler 8 is a utility that takes both the first template 1 and the second template 3 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first compiled template 2 and the second compiled template 4 in the Genesis Byte Code (GBC) 11 as output, respectively. Both the first compiled template 2 and the

second compiled template 4 contain the Genesis Byte Code (GBC) 11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte Code (GBC) 11.

Figure 2 describes a second embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a financial model 9 into a first template 21 in a Genesis Template Markup Language (GTML) 10 with a group of data 15 in assembly language (ASM) 13; a step of compiling the first template 21 into a first compiled template 22 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of creating a second template 23 in the Genesis Template Markup Language (GTML) 10 with a group of input data 12; a step of compiling the second template 23 into a second compiled template 24 in the Genesis Byte Code 11 through the Genesis Compiler 8; a step of interpreting the first compiled template 22 and the second compiled template 24 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 27 by the processor 19. Either one or both of the first template 21 or the second template 23 can be a single template or a group of templates. Either one or both of the first template 21 and the second template 23 may contain either text, or data, or instructions in the Genesis Template Markup Language (GTML) 10, or any combination of text, data, and instructions in the Genesis Template Markup Language (GTML) 10. Either one or both of the first compiled template 12 and the second compiled template 14 can be a single compiled template or a group of compiled templates. The Genesis Compiler 8 is a utility that takes both the first template 21 and the second template 23 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first

compiled template 22 and the second compiled template 24 in the Genesis Byte Code (GBC) 11 as output, respectively. Both the first compiled template 22 and the second compiled template 24 contain the Genesis Byte Code (GBC) 11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte Code (GBC) 11.

Figure 3 describes a third embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a financial model 9 into a first template 31 in a Genesis Template Markup Language (GTML) 10; a step of compiling the first template 31 into a first compiled template 32 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of creating a second template 33 with a group of input data 12 in the Genesis Template Markup Language (GTML) 10 and a group of data 16 in assembly language (ASM) 13; a step of compiling the second template 33 into a second compiled template 34 in the Genesis Byte Code 11 through the Genesis Compiler 8; a step of interpreting the first compiled template 32 and the second compiled template 34 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 37 by the processor 19. Either one or both of the first template 31 or the second template 33 can be a single template or a group of templates. Either one or both of the first template 31 and the second template 33 may contain either text, or data, or instructions in the Genesis Template Markup Language (GTML) 10, or any combination of text, data, and instructions in the Genesis Template Markup Language (GTML) 10. Either one or both of the first compiled template 12 and the second compiled template 14 can be a single compiled template or a group of compiled templates. The Genesis

Compiler 8 is a utility that takes both the first template 31 and the second template 33 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first compiled template 32 and the second compiled template 34 in the Genesis Byte Code (GBC) 11 as output, respectively. Both the first compiled template 32 and the second compiled template 34 contain the Genesis Byte Code (GBC) 11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte Code (GBC) 11.

Figure 4 describes a fourth embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a financial model 9 into a first template 41 in a Genesis Template Markup Language (GTML) 10 with a group of data 15 in assembly language (ASM) 13; a step of compiling the first template 41 into a first compiled template 42 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of creating a second template 43 in the Genesis Template Markup Language (GTML) 10 with a group of input data 12 and a group of data 16 in assembly language (ASM) 13; a step of compiling the second template 43 into a second compiled template 44 in the Genesis Byte Code 11 through the Genesis Compiler 8; a step of interpreting the first compiled template 42 and the second compiled template 44 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 47 by the processor 19. Either one or both of the first template 41 or the second template 43 can be a single template or a group of templates. Either one or both of the first template 41 and the second template 43 may contain either text, or data, or instructions in the Genesis Template Markup Language (GTML) 10, or any combination of text, data, and

instructions in the Genesis Template Markup Language (GTML) 10. Either one or both of the first compiled template 42 and the second compiled template 44 can be a single compiled template or a group of compiled templates. The Genesis Compiler 8 is a utility that takes both the first template 41 and the second template 43 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first compiled template 42 and the second compiled template 44 in the Genesis Byte Code (GBC) 11 as output, respectively. Both the first compiled template 42 and the second compiled template 44 contain the Genesis Byte Code (GBC) 11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte Code (GBC) 11.

Referring now to Figures 1 to 4, the Genesis Template Markup Language (GTML) 10 is linked to a high level programming language 15 through a Genesis Application Program Interface 16, the high level programming language 15 includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like. The Genesis Template Markup Language (GTML) 10 is a complete programming language with looping, branching, object definition, and like. The Genesis Template Markup Language features a rich set of instructions. The Genesis Byte Code (GBC) 11 is a binary representation of the instruction mnemonics created by the Genesis Compiler 8. The Genesis Application Program Interface 16 is a feature rich set of functions that allow a high level programming language 15 to pass data back and forth to a current execution of both the first compiled template 2, 22, 32, 42 and the second compiled template 4, 24, 34, 44.

Figure 5 describes a fifth embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in

templates. The method comprises a step of programming a financial model 9 into a first template 1 in C++ language 50; a step of compiling the first template 51 into a first compiled template 52 in a Genesis Byte Code (GBC) 11 through a Genesis C++ Compiler 58; a step of creating a second template 53 in Fortran language 55 with a group of input data 12; a step of compiling the second template 53 into a second compiled template 54 in the Genesis Byte Code 11 through the Genesis Fortran Compiler 59; a step of interpreting the first compiled template 52 and the second compiled template 54 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 57 by the processor 19. Either one or both of the first template 51 or the second template 53 can be a single template or a group of templates. The first template 51 may contain either text, or data, or instructions in the C++ language 50, or any combination of text, data, and instructions in the C++ language 50. The second template 53 may contain either text, or data, or instructions in the Fortran language 55, or any combination of text, data, and instructions in the Fortran language 55. Either one or both of the first compiled template 52 and the second compiled template 54 can be a single compiled template or a group of compiled templates. The Genesis C++ Compiler 58 is a utility that takes the first template 51 in the C++ language 50 as input, and produces the first compiled template 52 in the Genesis Byte Code (GBC) 11 as output. The Genesis Fortran Compiler 59 is a utility that takes the second template 53 in the Fortran language 55 as input, and produces the second compiled template 54 in the Genesis Byte Code (GBC) 11 as output. Both the first compiled template 52 and the second compiled template 54 contain the Genesis Byte Code (GBC) 11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte

Code (GBC) 11. The Genesis Byte Code (GBC) 11 is a binary representation of the instruction mnemonics created by the Genesis C++ Compiler 58 and the Genesis Fortran Compiler 59.

Referring now to Figures 1 to 5, the financial model 9 can be a single model or a group of models. The first compiled template 2, 22, 32, 42, 52 and the second compiled template 4, 24, 34, 44, 54 includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute the first compiled template 2, 22, 32, 42, 52 and the second compiled template 4, 24, 34, 44, 54, respectively. The first template 1, 21, 31, 41, 51 and the second template 3, 23, 33, 43, 53 are created independently from each other. The Genesis Byte Code (GBC) 11 is the lowest level form of code used by the Genesis Executor 5. The Genesis Byte Code (GBC) 11 consists of an ordered set of binary representations of instructions.

Figure 6 describes a sixth embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a financial model A 71 into a first template 61 in Genesis Template Markup Language (GTML) 10 with a group of data 76 in assembly language (ASM) 13; a step of compiling the first template 61 into a first compiled template 62 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of programming a financial model B 72 into a second template 63 in C++ language 50; a step of compiling the second template 63 into a second compiled template 64 in a Genesis Byte Code (GBC) 11 through a Genesis C++ Compiler 58; a step of programming a financial model C 73 into a third template 65 in Fortran language 55; a step of compiling the third template 65 into a third compiled template 66 in the Genesis

Byte Code 11 through the Genesis Fortran Compiler 59; a step of creating a fourth template 67 in the Genesis Template Markup Language (GTML) 10 with a first group of input data 74; a step of compiling the fourth template 67 into a fourth compiled template 68 in the Genesis Byte Code 11 through the Genesis Compiler 8; a step of creating a fifth template 69 in a C+ language 78 with a second group of input data 74; a step of compiling the fifth template 69 into a fifth compiled template 70 in the Genesis Byte Code 11 through the Genesis C+ Compiler 79; a step of interpreting the first compiled template 62, the second compiled template 64, the third compiled template 66, the fourth compiled template 68, and the fifth compiled template 70 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 77 by the processor 19. The Genesis Compiler 8 is a utility that takes both the first template 61 and the fourth template 67 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first compiled template 62 and the fourth compiled template 68 in the Genesis Byte Code (GBC) 11 as output, respectively. The Genesis C++ Compiler 58 is a utility that takes the second template 63 in the C++ language 50 as input, and produces the second compiled template 64 in the Genesis Byte Code (GBC) 11 as output. The Genesis Fortran Compiler 59 is a utility that takes the third template 65 in the Fortran language 55 as input, and produces the fourth compiled template 68 in the Genesis Byte Code (GBC) 11 as output. The Genesis C+ Compiler 79 is a utility that takes the fifth template 69 in the C+ language 78 as input, and produces the fifth compiled template 70 in the Genesis Byte Code (GBC) 11 as output. Each of the first compiled template 62, the second compiled template 64, the third compiled template 66, the fourth compiled template 68, and the fifth compiled template 70 contains the Genesis Byte Code (GBC)

11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte Code (GBC) 11. The Genesis Template Markup Language (GTML) 10 is linked to a high level programming language 15 through a Genesis Application Program Interface 16, the high level programming language 15 includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like. The Genesis Template Markup Language (GTML) 10 is a complete programming language with looping, branching, object definition, and like. The Genesis Template Markup Language features a rich set of instructions. The Genesis Byte Code (GBC) 11 is a binary representation of the instruction mnemonics created by the Genesis Compiler 8. The Genesis Application Program Interface 16 is a feature rich set of functions that allow a high level programming language 15 to pass data back and forth to a current execution of both the first compiled template 62 and the fourth compiled template 68.

Figure 7 describes a seventh embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a first group of instructions 91 into a first template 81 in Genesis Template Markup Language (GTML) 10 with a group of data 96 in assembly language (ASM) 13; a step of compiling the first template 81 into a first compiled template 82 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of programming a second group of instructions 92 into a second template 83 in C++ language 50; a step of compiling the second template 83 into a second compiled template 84 in a Genesis Byte Code (GBC) 11 through a Genesis C++ Compiler 58; a step of programming a third group of instructions 93 into a third template 85 in Fortran language 55; a step of compiling the third template 85 into a third compiled

template 86 in the Genesis Byte Code 11 through the Genesis Fortran Compiler 59; a step of creating a fourth template 87 in the Genesis Template Markup Language (GTML) 10 with a first group of input data 94; a step of compiling the fourth template 87 into a fourth compiled template 88 in the Genesis Byte Code 11 through the Genesis Compiler 8; a step of creating a fifth template 89 in a C+ language 78 with a second group of input data 94; a step of compiling the fifth template 89 into a fifth compiled template 90 in the Genesis Byte Code 11 through the Genesis C+ Compiler 79; a step of interpreting the first compiled template 82, the second compiled template 84, the third compiled template 86, the fourth compiled template 88, and the fifth compiled template 90 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 97 by the processor 19. The Genesis Compiler 8 is a utility that takes both the first template 81 and the fourth template 87 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first compiled template 82 and the fourth compiled template 88 in the Genesis Byte Code (GBC) 11 as output, respectively. The Genesis C++ Compiler 58 is a utility that takes the second template 83 in the C++ language 50 as input, and produces the second compiled template 84 in the Genesis Byte Code (GBC) 11 as output. The Genesis Fortran Compiler 59 is a utility that takes the third template 85 in the Fortran language 55 as input, and produces the fourth compiled template 88 in the Genesis Byte Code (GBC) 11 as output. The Genesis C+ Compiler 79 is a utility that takes the fifth template 89 in the C+ language 78 as input, and produces the fifth compiled template 90 in the Genesis Byte Code (GBC) 11 as output. Each of the first compiled template 82, the second compiled template 84, the third compiled template 86, the fourth compiled template 88, and the fifth compiled template 90 contains the Genesis Byte Code (GBC)

11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to variables, constants, and code required by the Genesis Byte Code (GBC) 11. The Genesis Template Markup Language (GTML) 10 is linked to a high level programming language 15 through a Genesis Application Program Interface 16. The Genesis Template Markup Language (GTML) 10 is a complete programming language with looping, branching, object definition, and like. The Genesis Template Markup Language (GTML) 10 features a rich set of instructions. The Genesis Byte Code (GBC) 11 is a binary representation of the instruction mnemonics created by the Genesis Compiler 8. The Genesis Application Program Interface 16 is a feature rich set of functions that allow the high level programming language 15 to pass data back and forth to a current execution of both the first compiled template 82 and the fourth compiled template 88. The high level programming language 15 includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

Figure 8 describes a eighth embodiment of a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates. The method comprises a step of programming a first group of instructions 111 into a first template 101 in Genesis Template Markup Language (GTML) 10 with a group of data 118 in assembly language (ASM) 13; a step of programming a second group of instructions 112 into a second template 102 in Genesis Template Markup Language (GTML) 10; a step of compiling the first template 101 and the second template 102 into a first compiled template 103 in a Genesis Byte Code (GBC) 11 through a Genesis Compiler 8; a step of creating a third template 104 in the Genesis Template Markup Language (GTML) 10 with a first group of input data 113; a step of creating a fourth

template 105 in the Genesis Template Markup Language (GTML) 10 with a second group of input data 114; a step of creating a fifth template 106 in the Genesis Template Markup Language (GTML) 10 with a third group of input data 115 and a group of data 119 in assembly language (ASM) 13; a step of compiling the third template 104, the fourth template 105 and the fifth template 106 into a second compiled template 107 in the Genesis Byte Code 11 through the Genesis Compiler 8; a step of creating a sixth template 108 in a C+ language 78 with a fourth group of input data 108; a step of creating a seventh template 109 in the Genesis Template Markup Language (GTML) 10 with a fifth group of input data 117; a step of compiling the sixth template 108 and the seventh template 109 into a third compiled template 110 in the Genesis Byte Code 11 through the Genesis C+ Compiler 79; a step of interpreting the first compiled template 103, the second compiled template 107 and the third compiled template 110 to a processor 19 by a Genesis Executor 5; and a step of generating a group of output data 117 by the processor 19. The Genesis Compiler 8 is a utility that takes the first template 101, the second template 102, the third template 104, the fourth template 105 and the fifth template 106 in the Genesis Template Markup Language (GTML) 10 as input, and produces the first compiled template 103 and the second compiled template 107 in the Genesis Byte Code (GBC) 11 as output, respectively. The Genesis C+ Compiler 79 is a utility that takes the sixth template 108 and the seventh template 109 in the C+ language 78 as input, and produces the third compiled template 110 in the Genesis Byte Code (GBC) 11 as output. Each of the first compiled template 103, the second compiled template 107 and the third compiled template 110 contains the Genesis Byte Code (GBC) 11 suitable to be interpreted by the Genesis Executor 5 to the processor 19, along with a set of maps to

variables, constants, and code required by the Genesis Byte Code (GBC) 11. The Genesis Template Markup Language (GTML) 10 is linked to a high level programming language 15 through a Genesis Application Program Interface 16, the high level programming language 15 includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like. The Genesis Template Markup Language (GTML) 10 is a complete programming language with looping, branching, object definition, and like. The Genesis Template Markup Language (GTML) 10 features a rich set of instructions. The Genesis Byte Code (GBC) 11 is a binary representation of the instruction mnemonics created by the Genesis Compiler 8. The Genesis Application Program Interface 16 is a feature rich set of functions that allow a high level programming language 15 to pass data back and forth to a current execution of both the first compiled template 103 and the fourth compiled template 107.

Referring now to Figures 1-8, the processor can be either a processor of a computer system or a processor of an embedded system.

Figure 9 provides a system on which this invention can be practiced. The system provides a system A 124, a first user interface 121 on system A 124, a Genesis Compiler 8 on system A 124, a Genesis Executor 5 on system A 124, a system B 126, a second user interface 122 on system B 126, and a Genesis C++ Compiler 58 on system B 126. A user can program a model 123 into a first template 131 in Genesis Template Markup Language (GTML) 10 through a first user interface 121 on system A 124. The Genesis Compiler 8 on system A 124 compiles the first template 131 into a first compiled template 127 in Genesis Byte Code (GBC) 11. Another user can create a second template 132 by using C++ language 50 with a group of data 125 through the second user interface

122 on system B 126. The Genesis C++ Compiler 58 on system B 126 compiles the second template 132 into a second compiled template 128 in Genesis Byte Code (GBC) 11. The second compiled template 128 is then sent to system A 124. The Genesis Executor 5 on system A 124 interprets both the first compiled template 127 and the second compiled template 128 to a processor 133 of system A 124. The processor 133 of system A 124 then generates a group of output data 129 on system A 124. The group of output data 129 is then sent to system B 126. The Genesis Template Markup Language (GTML) 10 is linked to a high level programming language 15 through a Genesis Application Program Interface (GAPI) 16. The Genesis Application Program Interface (GAPI) 16 is a feature rich set of functions that allow a high level programming language 15 to pass data back and forth to a current execution of both the first compiled template 127 and the second compiled template 128. The high level programming language 15 includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

Figure 10 provides another system on which this invention can be practiced. The system provides a system A 124, a first user interface 121 on system A 124, a Genesis Compiler 8 on system A 124, a system B 126, a second user interface 122 on system B 126, a Genesis C++ Compiler 58 on system B 126, and a Genesis Executor 5 on system B 126. A user can program a model 123 into a first template 131 in Genesis Template Markup Language (GTML) 10 through a first user interface 121 on system A 124. The Genesis Compiler 8 on system A 124 compiles the first template 131 into a first compiled template 127 in Genesis Byte Code (GBC) 11. The first compiled template 127 is then sent to system B 126. Another user can create a second template 132 by using C++ language 50 with a group of data 125 through the second user interface 122 on system B

126. The Genesis C++ Compiler 58 on system B 126 compiles the second template 132 into a second compiled template 128 in Genesis Byte Code (GBC) 11. The Genesis Executor 5 on system B 126 interprets both the first compiled template 127 and the second compiled template 128 to a processor 134 of system B 126. The processor 134 of system B 126 then generates a group of output data 129 on system B 126. The Genesis Template Markup Language (GTML) 10 is linked to a high level programming language 15 through a Genesis Application Program Interface (GAPI) 16. The Genesis Application Program Interface (GAPI) 16 is a feature rich set of functions that allow a high level programming language 15 to pass data back and forth to a current execution of both the first compiled template 127 and the second compiled template 128. The high level programming language 15 includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

Referring now to Figures 9 and 10, system A 124 and system B 126 can be either computer system or embedded system.

Hence, the present invention provides a method, apparatus and computer program product for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates.

The present invention also provides a method, apparatus and computer program product that separates an operating program into two independent parts, one is a model template, the other is an input data template.

The present invention further provides a method, apparatus and computer program product that enables a model to be used by the other parties without disclosure any of the content of the model.

The present invention still further provides a method, apparatus and computer program product that enables a programmer to program a model without a need to know the design of the user interface.

The present invention further provides a method, apparatus and computer program product that enables a user interface designer to design a user interface without a need to know the content of the model.

The present invention still further provides a method, apparatus and computer program product that uses Genesis Executor to integrate the model and the template to generate an output.

The present invention further provides a method, apparatus and computer program product that enables the separation of the processing and translation of data from the core device or application.

As various possible embodiments may be made in the above invention for use for different purposes and as various changes might be made in the embodiments and methods above set forth, it is understood that all of the above matters here set forth or shown in the accompanying drawings are to be interpreted as illustrative and not in a limiting sense.

I claim:

1. A method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said method comprising:

programming a financial model with instructions or a financial model with instructions and associated data into a first template in a Genesis Template Markup Language, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to a first high level programming language through a Genesis Application Program Interface, said first high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said first template in said Genesis Template Markup Language contains either text, or data, or instructions, or any combination of text, data, and instructions;

compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler, said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output, said first compiled template includes tables and maps of data, or variables, or instructions required to execute said first compiled template, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template;

creating a second template in said Genesis Template Markup Language with a group of input data or a group of input data with associated instructions, said Genesis Template Markup Language is a complete programming language with looping,

branching, object definition, and like, said Genesis Template Markup Language may be linked to a second high level programming language through said Genesis Application Program Interface, said second high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, and like, said second template in said Genesis Template Markup Language contains either text, or data, or instructions, or any combination of text, data, and instructions;

compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler, said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output, said second compiled template may include tables and maps of data, or variables, or instructions required to execute said second compiled template, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template;

interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

generating a group of output data by said processor.

2. The method in claim 1, wherein either one or both of said first template and said second template can include a group of data in assembly language.

3. The method in claim 1, wherein said financial model can be a single financial model or a group of financial models.

4. The method in claim 1, wherein either one or both of said first template and said second template can be a single template or a group of templates.

5. The method in claim 1, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

6. The method in claim 1, wherein said Genesis Template Markup Language features a rich set of instructions.

7. The method of claim 1, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

8. The method of claim 1, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

9. The method of claim 1, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

10. The method of claim 1, wherein said Genesis Application Program Interface is a feature rich set of functions that allow a high level programming language to pass data back and forth to a current execution of both said first compiled template and said second

compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

11. The method of claim 1, wherein said first template and said second template are created independently from each other.

12. A method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said method comprising:

programming a model with instructions into a first template in a Genesis Template Markup Language;

compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler;

creating a second template in said Genesis Template Markup Language with a group of input data;

compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler;

interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

generating a group of output data by said processor.

13. The method in claim 12, wherein said instructions can be either a group of instructions or a group of instructions with associated data.

14. The method in claim 12, wherein said group of input data can either be a group of data or a group of data with associated instructions.

15. The method in claim 12, wherein either one or both of said first template and said second template can include a group of data in assembly language.

16. The method in claim 12, wherein said Genesis Template Markup Language is linked to a high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like..

17. The method in claim 12, wherein said model can be a single model or a group of models.

18. The method in claim 12, wherein either one or both of said first template or said second template can be a single template or a group of templates.

19. The method in claim 12, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

20. The method in claim 12, wherein said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like.

21. The method in claim 12, wherein said Genesis Template Markup Language features a rich set of instructions.

22. The method in claim 12, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

23. The method in claim 12, wherein said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output.

24. The method in claim 12, wherein said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output.

25. The method in claim 12, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

26. The method in claim 12, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

27. The method in claim 12, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

28. The method of claim 12, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

29. The method of claim 12, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

30. The method of claim 12, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

31. The method of claim 12, wherein said model can be a single model or a group of models.

32. The method of claim 12, wherein said first template and said second template are created independently from each other.

33. A method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said method comprising:

programming a group of pre-arranged instructions into a first template in a Genesis Template Markup Language;

compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler;

creating a second template in said Genesis Template Markup Language with a group of input data;

compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler;

interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

generating a group of output data by said processor.

34. The method in claim 33, wherein said group of pre-arranged instructions can be either a group of pre-arranged instructions or a group of pre-arranged instructions with associated data.

35. The method in claim 33, wherein said group of input data can either be a group of data or a group of data with associated instructions.

36. The method in claim 33, wherein either one or both of said first template and said second template can include a group of data in assembly language.

37. The method in claim 33, wherein said Genesis Template Markup Language is linked to a high level programming language through a Genesis Application Program Interface, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

38. The method in claim 33, wherein either one or both of said first template and said second template can be a single template or a group of templates.

39. The method in claim 33, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

40. The method in claim 33, wherein said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like.

41. The method in claim 33, wherein said Genesis Template Markup Language features a rich set of instructions.

42. The method in claim 33, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

43. The method in claim 33, wherein said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output.

44. The method in claim 33, wherein said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output.

45. The method in claim 33, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

46. The method in claim 33, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

47. The method in claim 33, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by

said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

48. The method of claim 33, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

49. The method of claim 33, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

50. The method of claim 33, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

51. The method of claim 37, wherein said Genesis Application Program Interface is a feature rich set of functions that allow a high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

52. The method of claim 33, wherein said first template and said second template are created independently from each other.

53. A method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said method comprising:

programming a group of instructions into a first template, said group of instructions can be either a group of instructions or a group of instructions with associated data;

compiling said first template into a first compiled template in Genesis Byte Code through a first compiler;

creating a second template with a group of input data, said group of input data can either be a group of data or a group of data with associated instructions;

compiling said second template into a second compiled template in Genesis Byte Code through a second compiler;

interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

generating a group of output data by said processor.

54. The method in claim 53, wherein said first template is in a first high level programming language, said first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current execution of both said first compiled

template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

55. The method in claim 53, wherein said second template is in a second high level programming language, said second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

56. The method in claim 53, wherein either one or both of said first template and said second template can include a group of data in assembly language.

57. The method in claim 53, wherein either one or both of said first template and said second template can be a single template or a group of templates.

58. The method in claim 53, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

59. The method in claim 53, wherein either one or both of said first template and said second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

60. The method in claim 53, wherein said first compiler is a utility that takes said first template in said first high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

61. The method in claim 53, wherein said second compiler is a utility that takes said second template in said second high level programming language as input, and produces said second compiled template in said Genesis Byte Code as output.

62. The method in claim 53, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

63. The method in claim 53, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

64. The method in claim 53, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

65. The method of claim 53, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

66. The method of claim 53, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

67. The method of claim 53, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by either said first compiler or said second compiler.

68. The method of claim 53, wherein said first template and said second template are created independently from each other.

69. A method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said method comprising:

programming a group of instructions into a first template, said group of instructions can be either a group of instructions or a group of instructions with associated data;

compiling said first template into a first compiled template in Genesis Byte Code through a first compiler;

creating a second template with a group of input data, said group of input data can either be a group of data or a group of data with associated instructions;

compiling said second template into a second compiled template in Genesis Byte Code through a second compiler;

interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

generating a group of output data by said processor.

70. The method in claim 69, wherein said first template is in a first high level programming language, said first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current execution of both said first compiled

template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

71. The method in claim 70, wherein said first template can include a group of data in assembly language.

72. The method in claim 71, wherein said first template can be a single template or a group of templates.

73. The method in claim 72, wherein said first template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

74. The method of claim 73, wherein said first template is created independently from said second template.

75. The method in claim 74, wherein said second template can include a group of data in assembly language.

76. The method in claim 75, wherein said second template can be a single template or a group of templates.

77. The method in claim 76, wherein said second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

78. The method in claim 77, wherein said second template is in a second high level programming language, said second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

79. The method in claim 78, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

80. The method in claim 79, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

81. The method of claim 80, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

82. The method of claim 81, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

83. The method of claim 82, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by either said first compiler or said second compiler.

84. The method in claim 83, wherein said first compiler is a utility that takes said first template in said first high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

85. The method in claim 84, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

86. The method in claim 83, wherein said second compiler is a utility that takes said second template in said second high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

87. The method in claim 86, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

88. An apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said apparatus comprising:

means for programming a financial model with instructions or a financial model with instructions and associated data into a first template in a Genesis Template Markup Language, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to a first high level programming language through a Genesis Application Program Interface, said first high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said first template in said Genesis Template Markup Language contains either text, or data, or instructions, or any combination of text, data, and instructions;

means for compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler, said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output, said first compiled template includes tables and maps of data, or variables, or instructions required to execute said first compiled template, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template;

means for creating a second template in said Genesis Template Markup Language with a group of input data or a group of input data with associated instructions, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to a second high level programming language through said Genesis Application Program Interface, said second high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, and like, said second template in said Genesis Template Markup Language contains either text, or data, or instructions, or any combination of text, data, and instructions;

means for compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler, said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output, said second compiled template may include tables and maps of data, or variables, or instructions required to execute said second compiled template, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template;

means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

means for generating a group of output data by said processor.

89. The apparatus in claim 88, wherein either one or both of said first template and said second template can include a group of data in assembly language.

90. The apparatus in claim 88, wherein said financial model can be a single financial model or a group of financial models.

91. The apparatus in claim 88, wherein either one or both of said first template and said second template can be a single template or a group of templates.

92. The apparatus in claim 88, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

93. The apparatus in claim 88, wherein said Genesis Template Markup Language features a rich set of instructions.

94. The apparatus of claim 88, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

95. The apparatus of claim 88, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

96. The apparatus of claim 88, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

97. The apparatus of claim 88, wherein said Genesis Application Program Interface is a feature rich set of functions that allow high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

98. The apparatus of claim 88, wherein said first template and said second template are created independently from each other.

99. An apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said apparatus comprising:

means for programming a model with instructions into a first template in a Genesis Template Markup Language;

means for compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler;

means for creating a second template in said Genesis Template Markup Language with a group of input data;

means for compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler;

means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

means for generating a group of output data by said processor.

100. The apparatus in claim 99, wherein said instructions can be either a group of instructions or a group of instructions with associated data.

101. The apparatus in claim 99, wherein said group of input data can either be a group of data or a group of data with associated instructions.

102. The apparatus in claim 99, wherein either one or both of said first template and said second template can include a group of data in assembly language.

103. The apparatus in claim 99, wherein said Genesis Template Markup Language is linked to a high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

104. The apparatus in claim 99, wherein said model can be a single model or a group of models.

105. The apparatus in claim 99, wherein either one or both of said first template or said second template can be a single template or a group of templates.

106. The apparatus in claim 99, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

107. The apparatus in claim 99, wherein said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like.

108. The apparatus in claim 99, wherein said Genesis Template Markup Language features a rich set of instructions.

109. The apparatus in claim 99, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

110. The apparatus in claim 99, wherein said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output.

111. The apparatus in claim 99, wherein said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output.

112. The apparatus in claim 99, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

113. The apparatus in claim 99, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

114. The apparatus in claim 99, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

115. The apparatus of claim 99, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

116. The apparatus of claim 99, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

117. The apparatus of claim 99, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

118. The apparatus of claim 99, wherein said model can be a single model or a group of models.

119. The apparatus of claim 99, wherein said first template and said second template are created independently from each other.

120. An apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said apparatus comprising:

means for programming a group of pre-arranged instructions into a first template in a Genesis Template Markup Language;

means for compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler;

means for creating a second template in said Genesis Template Markup Language with a group of input data;

means for compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler;

means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

means for generating a group of output data by said processor.

121. The apparatus in claim 120, wherein said group of pre-arranged instructions can be either a group of pre-arranged instructions or a group of pre-arranged instructions with associated data.

122. The apparatus in claim 120, wherein said group of input data can either be a group of data or a group of data with associated instructions.

123. The apparatus in claim 120, wherein either one or both of said first template and said second template can include a group of data in assembly language.

124. The apparatus in claim 120, wherein said Genesis Template Markup Language is linked to a high level programming language through a Genesis Application Program Interface, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

125. The apparatus in claim 120, wherein either one or both of said first template and said second template can be a single template or a group of templates.

126. The apparatus in claim 120, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

127. The apparatus in claim 120, wherein said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like.

128. The apparatus in claim 120, wherein said Genesis Template Markup Language features a rich set of instructions.

129. The apparatus in claim 120, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

130. The apparatus in claim 120, wherein said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output.

131. The apparatus in claim 120, wherein said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output.

132. The apparatus in claim 120, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

133. The apparatus in claim 120, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and

maps of data, variables, and instructions required to execute said second compiled template.

134. The apparatus in claim 120, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

135. The apparatus of claim 120, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

136. The apparatus of claim 120, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

137. The apparatus of claim 120, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

138. The apparatus of claim 124, wherein said Genesis Application Program Interface is a feature rich set of functions that allow a high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

139. The apparatus of claim 120, wherein said first template and said second template are created independently from each other.

140. An apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said apparatus comprising:

means for programming a group of instructions into a first template, said group of instructions can be either a group of instructions or a group of instructions with associated data;

means for compiling said first template into a first compiled template in Genesis Byte Code through a first compiler;

means for creating a second template with a group of input data, said group of input data can either be a group of data or a group of data with associated instructions;

means for compiling said second template into a second compiled template in Genesis Byte Code through a second compiler;

means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

means for generating a group of output data by said processor.

141. The apparatus in claim 140, wherein said first template is in a first high level programming language, said first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis

Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

142. The apparatus in claim 140, wherein said second template is in a second high level programming language, said second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

143. The apparatus in claim 140, wherein either one or both of said first template and said second template can include a group of data in assembly language.

144. The apparatus in claim 140, wherein either one or both of said first template and said second template can be a single template or a group of templates.

145. The apparatus in claim 140, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

146. The apparatus in claim 140, wherein either one or both of said first template and said second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

147. The apparatus in claim 140, wherein said first compiler is a utility that takes said first template in said first high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

148. The apparatus in claim 140, wherein said second compiler is a utility that takes said second template in said second high level programming language as input, and produces said second compiled template in said Genesis Byte Code as output.

149. The apparatus in claim 140, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

150. The apparatus in claim 140, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

151. The apparatus in claim 140, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

152. The apparatus of claim 140, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

153. The apparatus of claim 140, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

154. The apparatus of claim 140, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by either said first compiler or said second compiler.

155. The apparatus of claim 140, wherein said first template and said second template are created independently from each other.

156. An apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said apparatus comprising:

means for programming a group of instructions into a first template, said group of instructions can be either a group of instructions or a group of instructions with associated data;

means for compiling said first template into a first compiled template in Genesis Byte Code through a first compiler;

means for creating a second template with a group of input data, said group of input data can either be a group of data or a group of data with associated instructions;

means for compiling said second template into a second compiled template in Genesis Byte Code through a second compiler;

means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

means for generating a group of output data by said processor.

157. The apparatus in claim 156, wherein said first template is in a first high level programming language, said first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming

language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

158. The apparatus in claim 157, wherein said first template can include a group of data in assembly language.

159. The apparatus in claim 158, wherein said first template can be a single template or a group of templates.

160. The apparatus in claim 159, wherein said first template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

161. The apparatus of claim 160, wherein said first template is created independently from said second template.

162. The apparatus in claim 161, wherein said second template can include a group of data in assembly language.

163. The apparatus in claim 162, wherein said second template can be a single template or a group of templates.

164. The apparatus in claim 163, wherein said second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

165. The apparatus in claim 164, wherein said second template is in a second high level programming language, said second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

166. The apparatus in claim 165, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

167. The apparatus in claim 166, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

168. The apparatus of claim 167, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

169. The apparatus of claim 168, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

170. The apparatus of claim 169, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by either said first compiler or said second compiler.

171. The apparatus in claim 170, wherein said first compiler is a utility that takes said first template in said first high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

172. The apparatus in claim 171, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

173. The apparatus in claim 170, wherein said second compiler is a utility that takes said second template in said second high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

174. The apparatus in claim 173, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

175. A computer program product recorded on a computer readable medium for a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said computer program product comprising:

computer readable means for programming a financial model with instructions or a financial model with instructions and associated data into a first template in a Genesis Template Markup Language, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to a first high level programming language through a Genesis Application Program Interface, said first high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said first template in said Genesis Template Markup Language contains either text, or data, or instructions, or any combination of text, data, and instructions;

computer readable means for compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler, said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output, said first compiled template includes tables and maps of data, or variables, or instructions required

to execute said first compiled template, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template;

computer readable means for creating a second template in said Genesis Template Markup Language with a group of input data or a group of input data with associated instructions, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to a second high level programming language through said Genesis Application Program Interface, said second high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, and like, said second template in said Genesis Template Markup Language contains either text, or data, or instructions, or any combination of text, data, and instructions;

computer readable means for compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler, said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output, said second compiled template may include tables and maps of data, or variables, or instructions required to execute said second compiled template, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template;

computer readable means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

computer readable means for generating a group of output data by said processor.

176. The computer program product in claim 175, wherein either one or both of said first template and said second template can include a group of data in assembly language.

177. The computer program product in claim 175, wherein said financial model can be a single financial model or a group of financial models.

178. The computer program product in claim 175, wherein either one or both of said first template and said second template can be a single template or a group of templates.

179. The computer program product in claim 175, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

180. The computer program product in claim 175, wherein said Genesis Template Markup Language features a rich set of instructions.

181. The computer program product of claim 175, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

182. The computer program product of claim 175, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

183. The computer program product of claim 175, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

184. The computer program product of claim 175, wherein said Genesis Application Program Interface is a feature rich set of functions that allow a high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said higher level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

185. The computer program product of claim 175, wherein said first template and said second template are created independently from each other.

186. A computer program product recorded on a computer readable medium for a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said computer program product comprising:

programming a model with instructions into a first template in a Genesis Template Markup Language;

compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler;

creating a second template in said Genesis Template Markup Language with a group of input data;

compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler;

interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

generating a group of output data by said processor.

187. The computer program product in claim 186, wherein said instructions can be either a group of instructions or a group of instructions with associated data.

188. The computer program product in claim 186, wherein said group of input data can either be a group of data or a group of data with associated instructions.

189. The computer program product in claim 186, wherein either one or both of said first template and said second template can include a group of data in assembly language.

190. The computer program product in claim 186, wherein said Genesis Template Markup Language is linked to a high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

191. The computer program product in claim 186, wherein said model can be a single model or a group of models.

192. The computer program product in claim 186, wherein either one or both of said first template or said second template can be a single template or a group of templates.

193. The computer program product in claim 186, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

194. The computer program product in claim 186, wherein said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like.

195. The computer program product in claim 186, wherein said Genesis Template Markup Language features a rich set of instructions.

196. The computer program product in claim 186, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

197. The computer program product in claim 186, wherein said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output.

198. The computer program product in claim 186, wherein said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output.

199. The computer program product in claim 186, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

200. The computer program product in claim 186, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

201. The computer program product in claim 186, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

202. The computer program product of claim 186, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

203. The computer program product of claim 186, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

204. The computer program product of claim 186, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

205. The computer program product of claim 186, wherein said model can be a single model or a group of models.

206. The computer program product of claim 186, wherein said first template and said second template are created independently from each other.

207. A computer program product recorded on a computer readable medium for a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said computer program product comprising:

a computer readable means for programming a group of pre-arranged instructions into a first template in a Genesis Template Markup Language;

a computer readable means for compiling said first template into a first compiled template in a Genesis Byte Code through a Genesis Compiler;

a computer readable means for creating a second template in said Genesis Template Markup Language with a group of input data;

a computer readable means for compiling said second template into a second compiled template in said Genesis Byte Code through said Genesis Compiler;

a computer readable means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

a computer readable means for generating a group of output data by said processor.

208. The computer program product in claim 207, wherein said group of pre-arranged instructions can be either a group of pre-arranged instructions or a group of pre-arranged instructions with associated data.

209. The computer program product in claim 207, wherein said group of input data can either be a group of data or a group of data with associated instructions.

210. The computer program product in claim 207, wherein either one or both of said first template and said second template can include a group of data in assembly language.

211. The computer program product in claim 207, wherein said Genesis Template Markup Language is linked to a high level programming language through a Genesis Application Program Interface, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

212. The computer program product in claim 207, wherein either one or both of said first template and said second template can be a single template or a group of templates.

213. The computer program product in claim 207, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

214. The computer program product in claim 207, wherein said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like.

215. The computer program product in claim 207, wherein said Genesis Template Markup Language features a rich set of instructions.

216. The computer program product in claim 207, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

217. The computer program product in claim 207, wherein said Genesis Compiler is a utility that takes said first template in said Genesis Template Markup Language as input, and produces said first compiled template in said Genesis Byte Code as output.

218. The computer program product in claim 207, wherein said Genesis Compiler is a utility that takes said second template in said Genesis Template Markup Language as input, and produces said second compiled template in said Genesis Byte Code as output.

219. The computer program product in claim 207, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

220. The computer program product in claim 207, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

221. The computer program product in claim 207, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

222. The computer program product of claim 207, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

223. The computer program product of claim 207, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

224. The computer program product of claim 207, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

225. The computer program product of claim 211, wherein said Genesis Application Program Interface is a feature rich set of functions that allow a high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

226. The computer program product of claim 207, wherein said first template and said second template are created independently from each other.

227. A computer program product recorded on a computer readable medium for a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said computer program product comprising:

computer readable means for programming a group of instructions into a first template, said group of instructions can be either a group of instructions or a group of instructions with associated data;

computer readable means for compiling said first template into a first compiled template in Genesis Byte Code through a first compiler;

computer readable means for creating a second template with a group of input data, said group of input data can either be a group of data or a group of data with associated instructions;

computer readable means for compiling said second template into a second compiled template in Genesis Byte Code through a second compiler;

computer readable means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

computer readable means for generating a group of output data by said processor.

228. The computer program product in claim 227, wherein said first template is in a first high level programming language, said first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

229. The computer program product in claim 227, wherein said second template is in a second high level programming language, said second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

230. The computer program product in claim 227, wherein either one or both of said first template and said second template can include a group of data in assembly language.

231. The computer program product in claim 227, wherein either one or both of said first template and said second template can be a single template or a group of templates.

232. The computer program product in claim 227, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

233. The computer program product in claim 227, wherein either one or both of said first template and said second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

234. The computer program product in claim 227, wherein said first compiler is a utility that takes said first template in said first high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

235. The computer program product in claim 227, wherein said second compiler is a utility that takes said second template in said second high level programming language as input, and produces said second compiled template in said Genesis Byte Code as output.

236. The computer program product in claim 227, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

237. The computer program product in claim 227, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

238. The computer program product in claim 227, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

239. The computer program product of claim 227, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

240. The computer program product of claim 227, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

241. The computer program product of claim 227, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by either said first compiler or said second compiler.

242. The computer program product of claim 227, wherein said first template and said second template are created independently from each other.

243. A computer program product recorded on a computer readable medium for a method for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said computer program product comprising:

computer readable means for programming a group of instructions into a first template, said group of instructions can be either a group of instructions or a group of instructions with associated data;

computer readable means for compiling said first template into a first compiled template in Genesis Byte Code through a first compiler;

computer readable means for creating a second template with a group of input data, said group of input data can either be a group of data or a group of data with associated instructions;

computer readable means for compiling said second template into a second compiled template in Genesis Byte Code through a second compiler;

computer readable means for interpreting said first compiled template and said second compiled template to a processor by a Genesis Executor; and

computer readable means for generating a group of output data by said processor.

244. The computer program product in claim 243, wherein said first template is in a first high level programming language, said first high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current

execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

245. The computer program product in claim 244, wherein said first template can include a group of data in assembly language.

246. The computer program product in claim 245, wherein said first template can be a single template or a group of templates.

247. The computer program product in claim 246, wherein said first template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

248. The computer program product of claim 247, wherein said first template is created independently from said second template.

249. The computer program product in claim 248, wherein said second template can include a group of data in assembly language.

250. The computer program product in claim 249, wherein said second template can be a single template or a group of templates.

251. The computer program product in claim 250, wherein said second template may contain either text, or data, or instructions, or any combination of text, data, and instructions.

252. The computer program product in claim 251, wherein said second template is in a second high level programming language, said second high level programming language includes Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language features a rich set of instructions and is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

253. The computer program product in claim 252, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

254. The computer program product in claim 253, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to

be interpreted by said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

255. The computer program product of claim 254, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

256. The computer program product of claim 255, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

257. The computer program product of claim 256, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by either said first compiler or said second compiler.

258. The computer program product in claim 257, wherein said first compiler is a utility that takes said first template in said first high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

259. The computer program product in claim 258, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

260. The computer program product in claim 257, wherein said second compiler is a utility that takes said second template in said second high level programming language as input, and produces said first compiled template in said Genesis Byte Code as output.

261. The computer program product in claim 260, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

262. An apparatus for translating data streams and performing programmatic tasks by using instructions and associated data stored in templates, said apparatus comprising:

a first compiler on a first system capable of compiling a first template in a first high level programming language into a first compiled template in a Genesis Byte Code;

a second compiler on a second system capable of compiling a second template in a second high level programming language into a second compiled template in said Genesis Byte Code;

a Genesis Executor on a third system capable of providing an interpretation of said first compiled template and said second compiled template; and

a processor on a forth system capable of generating a group of output data based on said interpretation of said first compiled template and said second compiled template by said Genesis Executor.

263. The apparatus in claim 262, wherein said first system, said second system, said third system and said fourth system are connected to each other either directly or through a system network.

264. The apparatus in claim 262, wherein said first system, said second system, said third system and said fourth system can be same system or different systems.

265. The apparatus in claim 262, wherein said first compiler and said second compiler can be same compiler or two different compilers.

266. The apparatus in claim 262, wherein said first template includes a group of pre-arranged instructions, said group of pre-arranged instructions can be either a group of pre-arranged instructions or a group of pre-arranged instructions with associated data.

267. The apparatus in claim 262, wherein said second template includes a group of input data, said group of input data can either be a group of data or a group of data with associated instructions.

268. The apparatus in claim 262, wherein either one or both of said first template and said second template can include a group of data in assembly language.

269. The apparatus in claim 262, wherein said first high level programming language can be Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN,

COBOL, or like, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language features a rich set of instructions, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language includes C, C+, C++, PASCAL, FORTRAN, COBOL, or like.

270. The apparatus in claim 262, wherein said second high level programming language can be Genesis Template Markup Language, C, C+, C++, PASCAL, FORTRAN, COBOL, or like, said Genesis Template Markup Language is a complete programming language with looping, branching, object definition, and like, said Genesis Template Markup Language features a rich set of instructions, said Genesis Template Markup Language may be linked to another high level programming language through a Genesis Application Program Interface, said Genesis Application Program Interface is a feature rich set of functions that allow said another high level programming language to pass data back and forth to a current execution of both said first compiled template and said second compiled template, said another high level programming language can be C, C+, C++, PASCAL, FORTRAN, COBOL, or like, .

271. The apparatus in claim 262, wherein either one or both of said first template and said second template can be a single template or a group of templates.

272. The apparatus in claim 262, wherein either one or both of said first compiled template and said second compiled template can be a single compiled template or a group of compiled templates.

273. The apparatus in claim 262, wherein either one or both of said first template and said second template may contain either text, or data, or instructions in said Genesis Template Markup Language, or any combination of text, data, and instructions in said Genesis Template Markup Language.

274. The apparatus in claim 262, wherein said first compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said first compiled template.

275. The apparatus in claim 262, wherein said second compiled template includes tables and maps of data, or variables, or instructions, or any combination of tables and maps of data, variables, and instructions required to execute said second compiled template.

276. The apparatus in claim 262, wherein both said first compiled template and said second compiled template contain said Genesis Byte Code suitable to be interpreted by

said Genesis Executor to said processor, along with a set of maps to variables, constants, and code required by said Genesis Byte Code.

277. The apparatus of claim 262, wherein said Genesis Byte Code is the lowest level form of code used by said Genesis Executor.

278. The apparatus of claim 262, wherein said Genesis Byte Code consists of an ordered set of binary representations of instructions.

279. The apparatus of claim 262, wherein said Genesis Byte Code is a binary representation of the instruction mnemonics created by said Genesis Compiler.

280. The apparatus of claim 262, wherein said first template and said second template are created independently from each other.

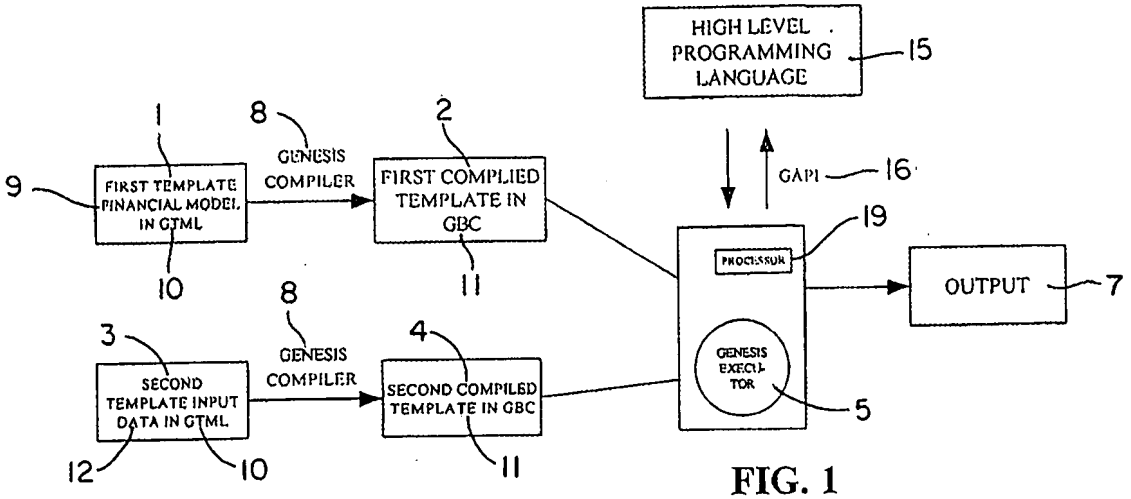

HIGH LEVEL PROGRAMMING LANGUAGE
15
GAPI
16
1
8
GENESIS COMPILER
2
9
FIRST TEMPLATE FINANCIAL MODEL IN GTML
FIRST COMPILED TEMPLATE IN GBC
10
11
19
PROCESSOR
OUTPUT
7
8
3
GENESIS COMPILER
4
SECOND TEMPLATE INPUT DATA IN GTML
SECOND COMPILED TEMPLATE IN GBC
GENESIS EXECUTOR
5
12
10
11

FIG. 1

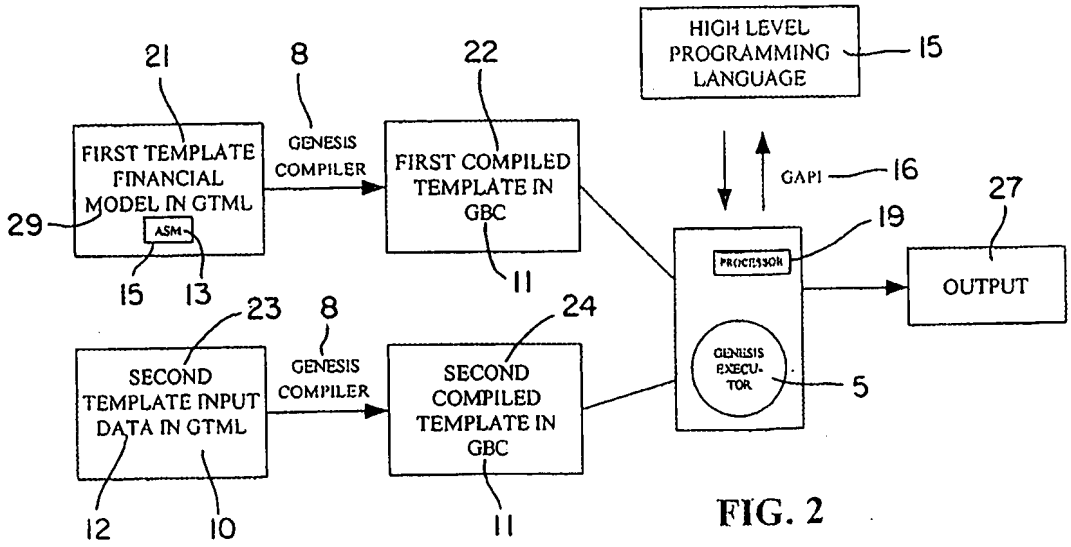

HIGH LEVEL PROGRAMMING LANGUAGE
15
21
8
22
GENESIS COMPILER
FIRST TEMPLATE FINANCIAL MODEL IN GTML
FIRST COMPILED TEMPLATE IN GBC
GAPI
16
27
29
ASM
19
PROCESSOR
OUTPUT
15
13
11
23
8
24
SECOND TEMPLATE INPUT DATA IN GTML
GENESIS COMPILER
SECOND COMPILED TEMPLATE IN GBC
GENESIS EXECUTOR
5
12
10
11

FIG. 2

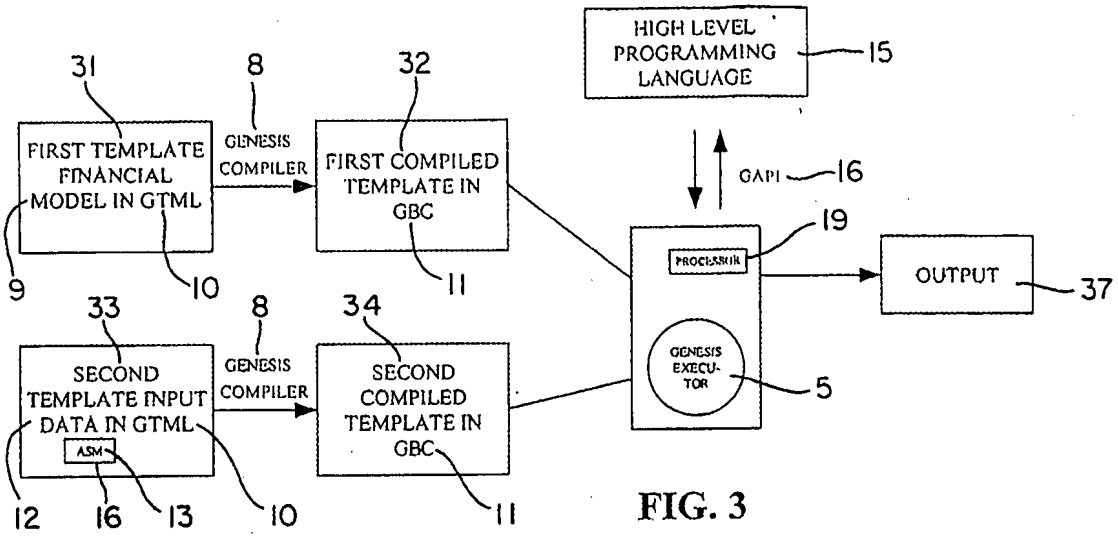

HIGH LEVEL PROGRAMMING LANGUAGE
15
31
8
32
FIRST TEMPLATE FINANCIAL MODEL IN GTML
GENESIS COMPILER
FIRST COMPILED TEMPLATE IN GBC
GAPI
16
19
PROCESSOR
OUTPUT
37
9
10
11
33
8
34
SECOND TEMPLATE INPUT DATA IN GTML
GENESIS COMPILER
SECOND COMPILED TEMPLATE IN GBC
GENESIS EXECU-TOR
5
ASM
12
16
13
10
11

FIG. 3

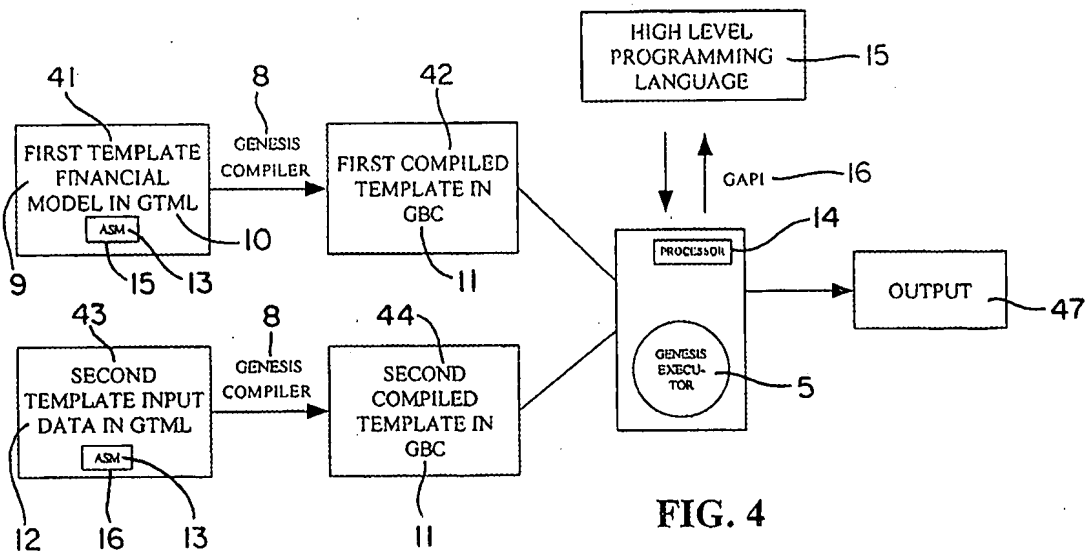

HIGH LEVEL PROGRAMMING LANGUAGE
15
41
8
42
FIRST TEMPLATE FINANCIAL MODEL IN GTML
GENESIS COMPILER
FIRST COMPILED TEMPLATE IN GBC
GAPI
16
14
PROCESSOR
OUTPUT
47
ASM
9
10
15
13
11
43
8
44
SECOND TEMPLATE INPUT DATA IN GTML
GENESIS COMPILER
SECOND COMPILED TEMPLATE IN GBC
GENESIS EXECU-TOR
5
ASM
12
16
13
11

FIG. 4

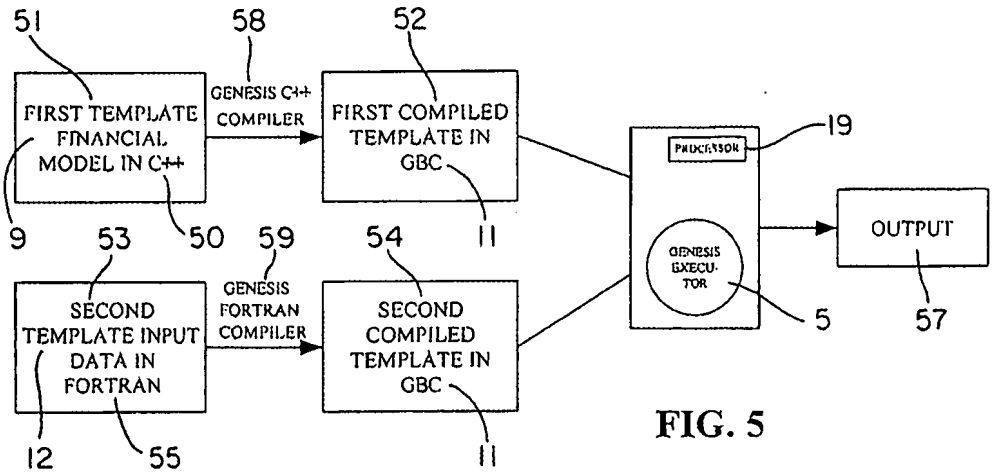

51
58
52
FIRST TEMPLATE FINANCIAL MODEL IN C++
GENESIS C++ COMPILER
FIRST COMPILED TEMPLATE IN GBC
19
PROCESSOR
9
53
50
59
54
11
GENESIS EXECU-TOR
OUTPUT
SECOND TEMPLATE INPUT DATA IN FORTRAN
GENESIS FORTRAN COMPILER
SECOND COMPILED TEMPLATE IN GBC
5
57
12
55
11

FIG. 5

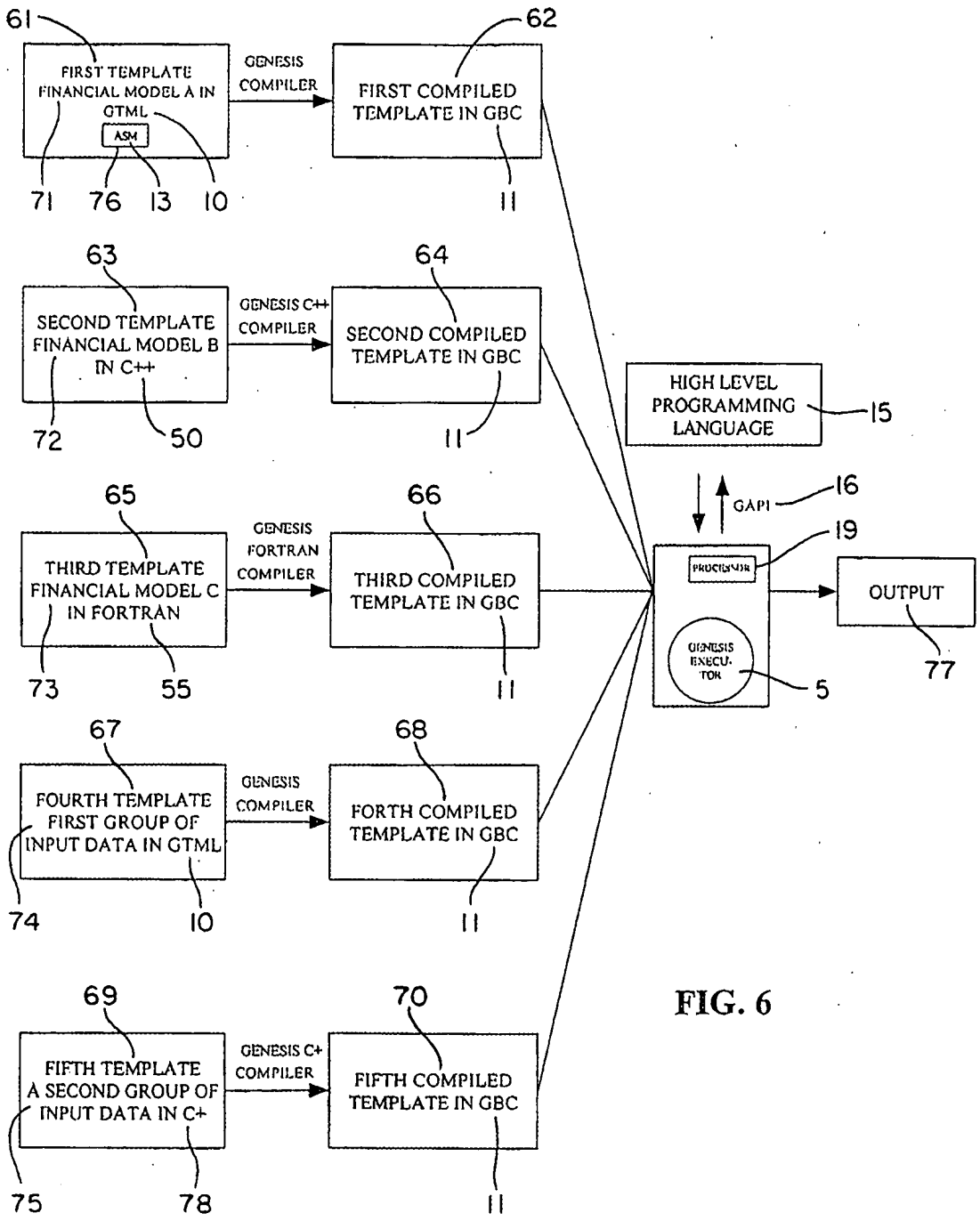

61
FIRST TEMPLATE FINANCIAL MODEL A IN GTML
ASM
71
76
13
10
GENESIS COMPILER
62
FIRST COMPILED TEMPLATE IN GBC
11
63
SECOND TEMPLATE FINANCIAL MODEL B IN C++
72
50
GENESIS C++ COMPILER
64
SECOND COMPILED TEMPLATE IN GBC
11
65
THIRD TEMPLATE FINANCIAL MODEL C IN FORTRAN
73
55
GENESIS FORTRAN COMPILER
66
THIRD COMPILED TEMPLATE IN GBC
11
67
FOURTH TEMPLATE FIRST GROUP OF INPUT DATA IN GTML
74
10
GENESIS COMPILER
68
FORTH COMPILED TEMPLATE IN GBC
11
69
FIFTH TEMPLATE A SECOND GROUP OF INPUT DATA IN C+
75
78
GENESIS C+ COMPILER
70
FIFTH COMPILED TEMPLATE IN GBC
11
HIGH LEVEL PROGRAMMING LANGUAGE
15
GAPI
16
19
PROCESSOR
GENESIS EXECUTOR
5
OUTPUT
77

FIG. 6

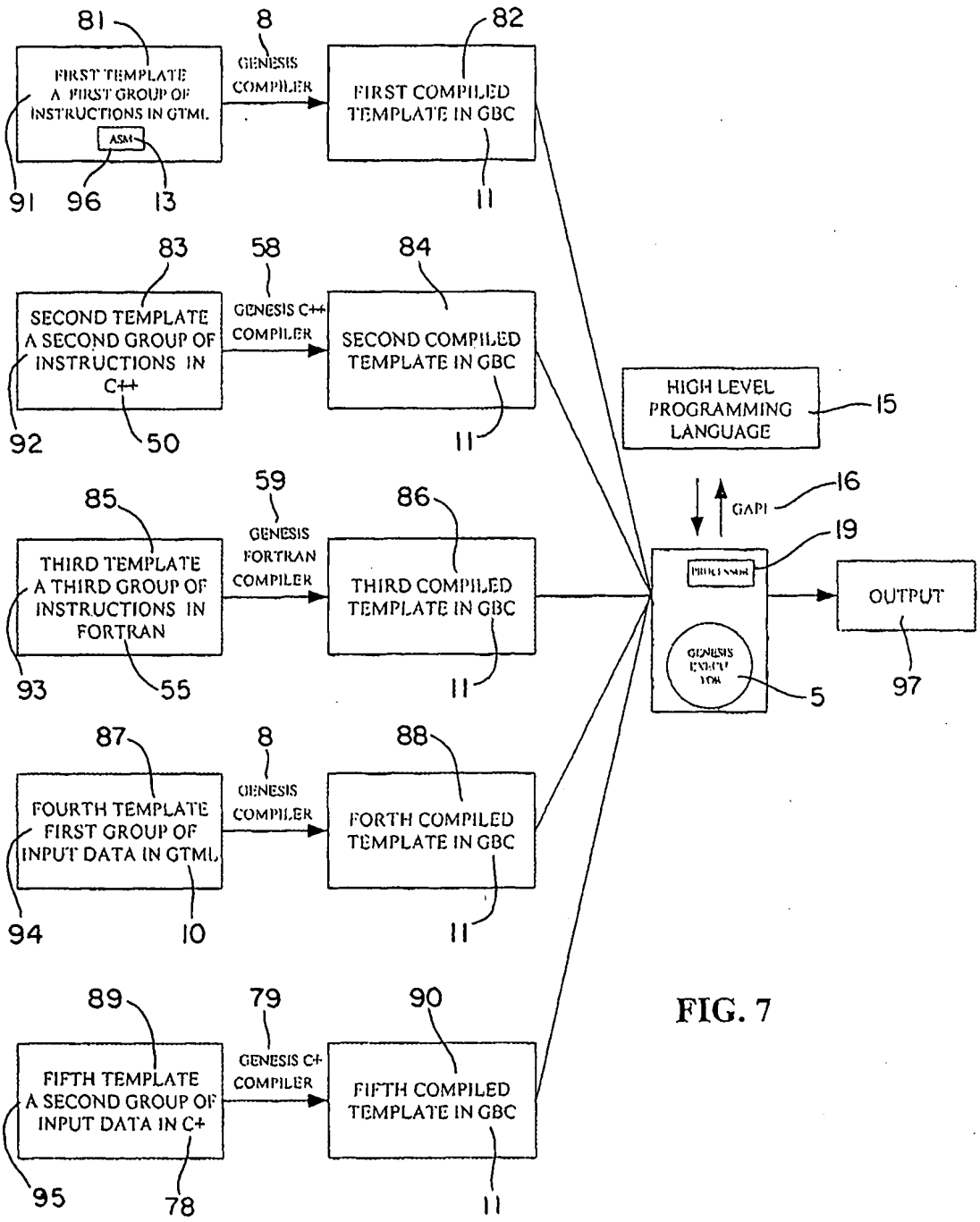

81
FIRST TEMPLATE A FIRST GROUP OF INSTRUCTIONS IN GTML
ASM
91
96
13
8
GENESIS COMPILER
82
FIRST COMPILED TEMPLATE IN GBC
11
83
SECOND TEMPLATE A SECOND GROUP OF INSTRUCTIONS IN C++
92
50
58
GENESIS C++ COMPILER
84
SECOND COMPILED TEMPLATE IN GBC
11
85
THIRD TEMPLATE A THIRD GROUP OF INSTRUCTIONS IN FORTRAN
93
55
59
GENESIS FORTRAN COMPILER
86
THIRD COMPILED TEMPLATE IN GBC
11
87
FOURTH TEMPLATE FIRST GROUP OF INPUT DATA IN GTML
94
10
8
GENESIS COMPILER
88
FORTH COMPILED TEMPLATE IN GBC
11
89
FIFTH TEMPLATE A SECOND GROUP OF INPUT DATA IN C+
95
78
79
GENESIS C+ COMPILER
90
FIFTH COMPILED TEMPLATE IN GBC
11
HIGH LEVEL PROGRAMMING LANGUAGE
15
GAPI
16
19
OUTPUT
97
5

FIG. 7

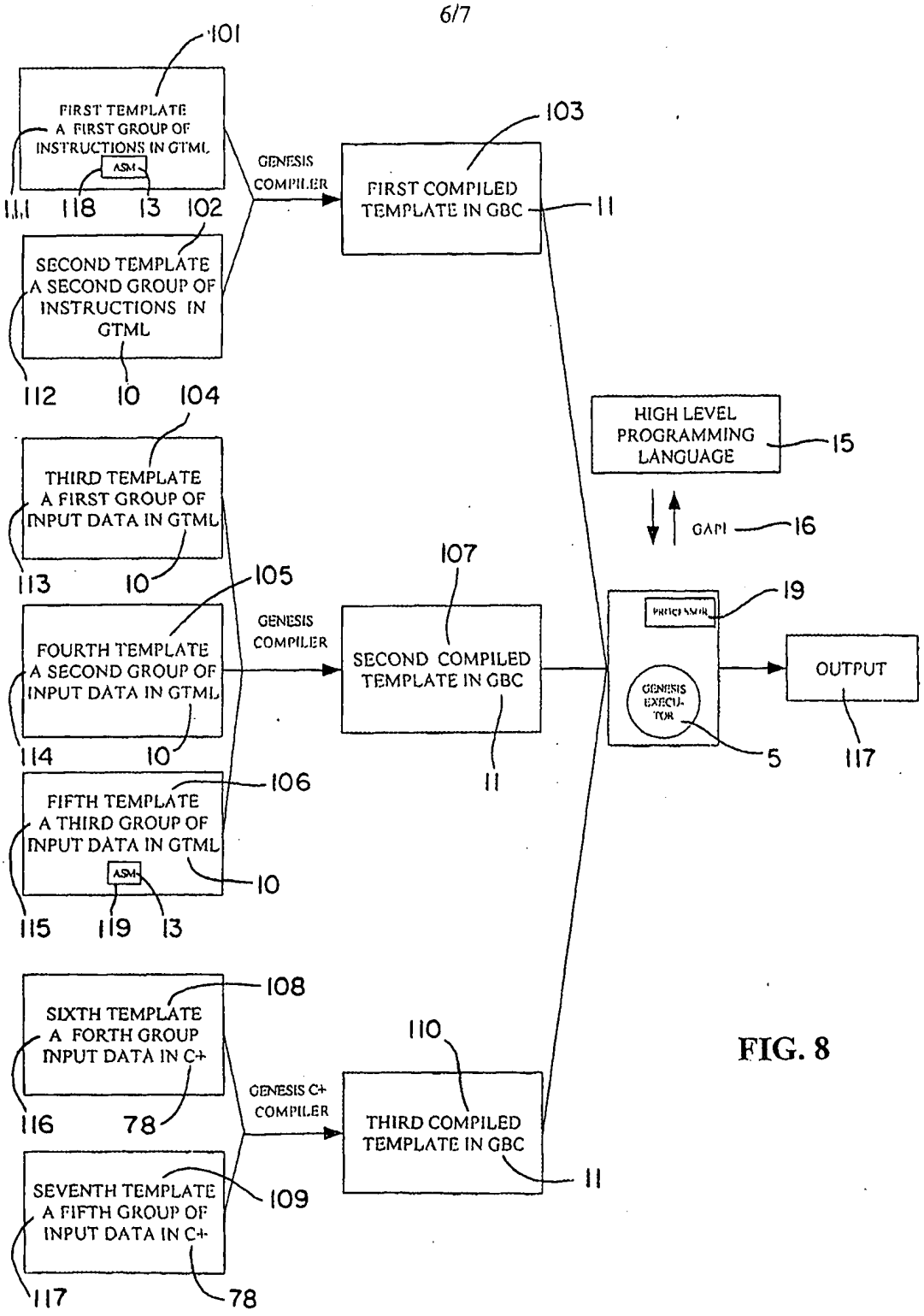

101
FIRST TEMPLATE A FIRST GROUP OF INSTRUCTIONS IN GTML
ASM
111
118
13
102
SECOND TEMPLATE A SECOND GROUP OF INSTRUCTIONS IN GTML
112
10
GENESIS COMPILER
103
FIRST COMPILED TEMPLATE IN GBC
11
HIGH LEVEL PROGRAMMING LANGUAGE
15
GAPI
16
104
THIRD TEMPLATE A FIRST GROUP OF INPUT DATA IN GTML
113
10
105
FOURTH TEMPLATE A SECOND GROUP OF INPUT DATA IN GTML
114
10
GENESIS COMPILER
107
SECOND COMPILED TEMPLATE IN GBC
11
19
OUTPUT
GENESIS EXECU-TOR
5
117
106
FIFTH TEMPLATE A THIRD GROUP OF INPUT DATA IN GTML
ASM
10
115
119
13
108
SIXTH TEMPLATE A FORTH GROUP INPUT DATA IN C+
116
78
GENESIS C+ COMPILER
110
THIRD COMPILED TEMPLATE IN GBC
11
109
SEVENTH TEMPLATE A FIFTH GROUP OF INPUT DATA IN C+
117
78

FIG. 8

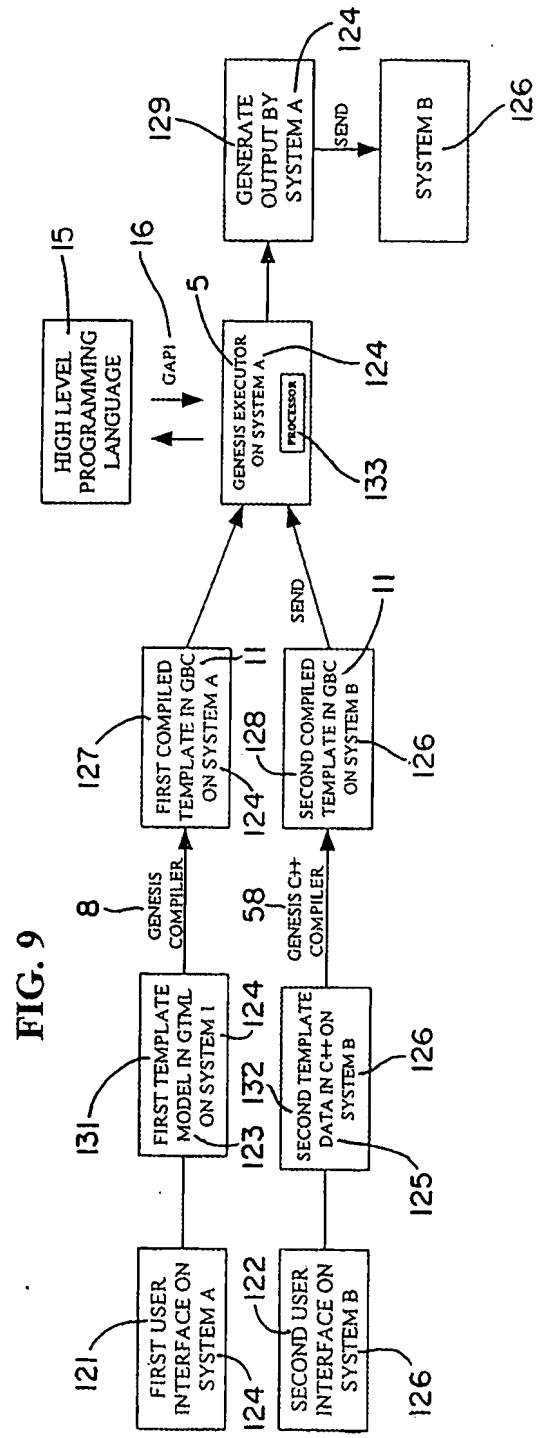

FIRST USER INTERFACE ON SYSTEM A
SECOND USER INTERFACE ON SYSTEM B
FIRST TEMPLATE MODEL IN GTML ON SYSTEM I
SECOND TEMPLATE DATA IN C++ ON SYSTEM B
GENESIS COMPILER
GENESIS C++ COMPILER
FIRST COMPILED TEMPLATE IN GBC ON SYSTEM A
SECOND COMPILED TEMPLATE IN GBC ON SYSTEM B
SEND
HIGH LEVEL PROGRAMMING LANGUAGE
GAPI
GENESIS EXECUTOR ON SYSTEM A
PROCESSOR
GENERATE OUTPUT BY SYSTEM A
SEND
SYSTEM B

FIG. 9

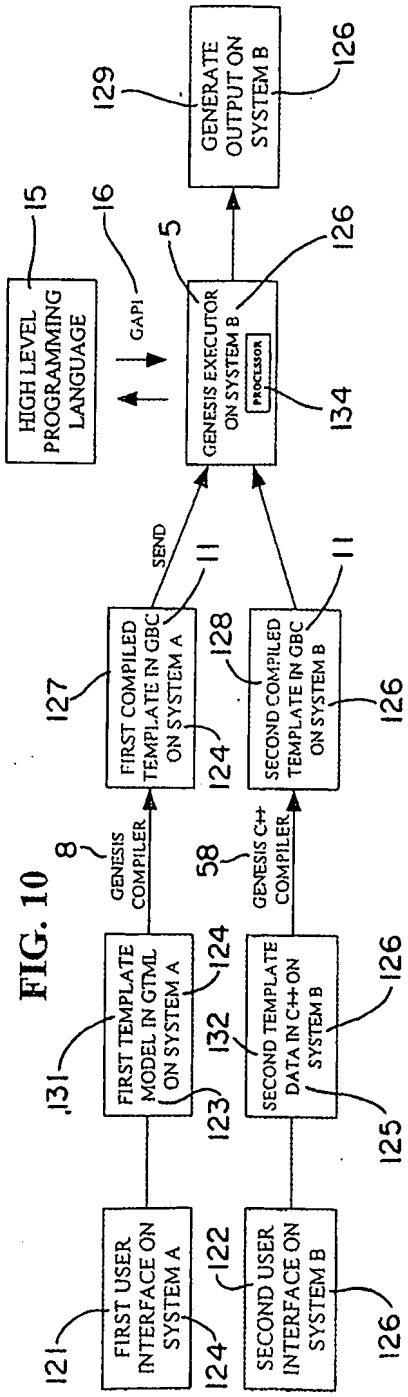

FIRST USER INTERFACE ON SYSTEM A
SECOND USER INTERFACE ON SYSTEM B
FIRST TEMPLATE MODEL IN GTML ON SYSTEM A
SECOND TEMPLATE DATA IN C++ ON SYSTEM B
GENESIS COMPILER
GENESIS C++ COMPILER
FIRST COMPILED TEMPLATE IN GBC ON SYSTEM A
SECOND COMPILED TEMPLATE IN GBC ON SYSTEM B
SEND
HIGH LEVEL PROGRAMMING LANGUAGE
GAPI
GENESIS EXECUTOR ON SYSTEM B
PROCESSOR
GENERATE OUTPUT ON SYSTEM B

FIG. 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/31713

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 9/45
US CL : 717/139; 717/140
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 717/139; 717/140; 712/208; 712/209; 712/210; 712/211; 712/212; 712/213

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
THE INTERNET

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WEST (USPAT, PG PUBS, DERWENT, IBM TDB, EPO, JPO)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	CHEN, J., GENESIS PRESENTATION, P. 1-14.	1-280
Y	US 6,108,696 A (MENDHEKAR ET AL.) 22 AUGUST 2000, FIGS. 1-3, COLS. 1-6 AND 9-11.	1-280
Y,P	US 2001/0032254 A1 (HAWKINS, J.) 18 OCTOBER 2001, FIGS 1,2,6,14, AND PAGE 5 2ND COL.	1-280
A,P	US 6,253,326 B1 (LINCKE ET AL.) 26 JUNE 2001, FIGS 1,2,4	1-280
Y	ZAROWIN ET AL, FINALLY, BUSINESS TALKS THE SAME LANGUAGE, AUGUST 2000, P. 1-8	1-280
A	US 6,058,373 A (BLINN ET AL.) 02 MAY 2000, FIGS 1-4	1-280

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance, the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance, the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 12 DECEMBER 2001	Date of mailing of the international search report 09 January 2002 (09.01.02)
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer KEVIN TESKA James R. Matthews Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/31713

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,930,341 A (CARDILLO, IV ET AL) 27 JULY 1999, FIGS 1-2, COLS 2-3, AND 6-7	1-280
A	US 6,006,242 A (POOLE ET AL.) 21 DECEMBER 1999, FIGS 7-10 AND COLS 3-6.	1-280